George Sono
CSE20
Prof. Hari. K.
Assignment 10.1

Car Class (parent)

Description:

Constructor:

This is a basic parent class which has the basic outline of a car. This means that its data variables are the car's name (self.__car_name), the model of the car (self.__model), the owner of the car (self.__car_name) and the manufacturer of the car (self.__manufacturer).

Methods:

- get_car_name(self):
    - This get method returns the name of the car (defined when creating instance) eg. instance.get_car_name() outputs instance.car_name
- get_model(self)
    - This get method returns the model of the car (defined when creating instance) eg. instance.get_model() returns instance.model
- get_owner(self)
    - This get method returns the owner of the car (defined when creating instance) eg. instance.get_owner() returns instance.owner
- get_manufacturer(self)
    - This get method returns the manufacturer of the car (defined when creating instance) eg. instance.get_manufacturer() returns instance.manufacturer
- __str__(self)
    - This magic method returns a description of the car (using the instance's data variables) eg. str(instance) returns "{instance.car_name} is owned by {instance.owner}. {instance.car_name} is manufactured by {instance.car_manfacturer} and its model is the {instance.model}."

CombustionEngineCar Class (child of Car)

Constructor:

As CombustionEngineCar Class is a child class of the parent class Car, it inherits the data variables of car_name, model, owner and manufacturer. With the super() function, it's able to use the get methods mentioned in the Car class method description. The new parameters to create an instance of this class are: cylinders, engine_layout, octane_rating, air_intake, exhaust, fuel_injection. However, these extra parameters are by default set to an average vehicle (specifically a Honda Civic).

Class attributes:

- uses_gas = True
    - This is to verify that it uses gas or not (in the ElectricCar class this is False)

Methods:

- add_turbo(self)
    - This updates the air_intake to a turbo-charger. In real life, a turbo increases the density of air entering the engine. It works by having the exhaust spinning a fan which is connected to the intake. With more air exiting the exhaust, the higher rpms the intake fan spins, and thus denser air is forced in. With denser air, the combustion within the engine is greater, thus leading to greater power (horsepower). What the method does is that it sets the instance's air_intake attribute to "turbo-charger" from "stock" (if air_intake was not defined in the constructor)
- add_supercharger(self)
    - This updates the air_intake attribute to a supercharger. Similar to a turbocharger, a supercharger is spun by the engine instead of the rate of air exiting the exhaust. This means that whenever the engine is running, the supercharger is on and always forcing denser air to enter the engine. This allows for an increase in horsepower also. What the method does to the class is that it sets the instance's air_intake attribute to "super-charger" from "stock" (if air_intake was not defined in the constructor)
- upgrade_fuel_injection(self)
    - This upgrades the fuel-injection system and updates fuel injection attributes from "stock" to "aftermarket". In real life, better fuel injection leads to better combustion within the engine and thus better performance (horsepower).
- upgrade_exhaust(self)
    - This upgrades the exhaust system and updates the exhaust attribute from "stock" to "aftermarket". In real life, a larger exhaust system allows for more air to exit the engine, allowing for it to take in more air. Theoretically this should lead to a gain in performance (horsepower).
- engine(self)
    - This method when called returns a dictionary regarding the information of the engine. More specifically, it returns a dictionary in this format {"layout": instance.engine_layout, "cylinders": instance.cylinders}

- is_it_stock(self)
    - This method when called returns a boolean value based on the fuel_injection, air_intake and exhaust attributes. If all of these are "stock" then it will return True. Otherwise if one of these are not "stock" it will return False.
- get_horsepower(self)
    - This method when called returns the horsepower value of the car (at the engine). It is based on all the attributes that are given when defining the instance and methods called to update the instance's attributes. This means that it takes into consideration all the parameters in the constructor that aren't inherited from class Car (cylinders, engine layout, fuel injection, exhaust, air intake, octane rating). Eg. instance.get_horsepower outputs 120 (float).
- get_wheel_horsepower(self)
    - This method is very similar to the get_horsepower but measures the horsepower rating at the wheels. This is as there are energy losses that occur when transferring energy from the crankshaft of the engine to the wheels.
- quick_stats(self)
    - This method prints an overview of the car's important stats: the car's name, horsepower, wheel horsepower, and most importantly, if it is stock.

    - It calls the methods described above to obtain all these aforementioned values
- __str__(self)
    - The __str__ is reimplemented to compensate for the car having an internal combustion engine. Thus in this method it also mentions the other attributes of the car like how many cylinders it has, the engine layout, etc

ElectricCar Class (child of Car)

Constructor:
The Electric Car class - a child of class Car - inherits the car_name, model, owner and manufacturer parameters and methods with super(). Regarding the other attributes, motors, motor wattage and battery range, these are set to a default value of a Nissan Leaf if not defined when creating the instance. As electric vehicles compared to gasoline vehicles have considerably less moving parts and have less parts, I thought it would be fitting if it had less parameters compared to the CombustionEngine class.

Class attributes:
- runs_gas = False
    - Even though an instance is not created, if the class attribute of runs_gas is called, it will always return False (boolean)

Methods:

- get_horsepower(self)
    - This returns the value of horsepower value of the instance as an integer. To calculate horsepower it uses the motor wattage and the amount of motors the instance has.
- replace_motor(self,motor_tuple)
    - This method allows you to replace the motor of the instance. As power is also determined by the wattage of the motor as well, a tuple in the form of (# of motors, motor wattage) is required. Iterating through the tuple by index, it updates the instance.motors and instance.motor_wattage attributes respectively.
- powertrain(self)
    - Similar to CombustionEngineCar's engine method, it returns a dictionary containing information about the instance's motor count, motor wattage and battery range. It returns a dictionary in the format of {"motors":instance.motors, "wattage": instance.motor_wattage, "battery range": instance.battery_range}
- quick_stats(self)
    - This method prints an overview of the car using the methods mentioned above. It prints information regarding the name of the vehicle, its horsepower, how many motors it has, the motor wattage and lastly its battery range.
- __str__(self)
    - The __str__ is reimplemented to compensate for the car being an electric vehicle. Thus in this method it also mentions the other attributes of the car like how many motors it has, engine wattage, range, etc

Dragstrip

Description:

With the DragStrip class, you can test and race your ElectricCar and CombustionEngineCar instances you've built.

Class attributes:

- material = "tarmac"
    - This is to show the road material of Dragstrip and also the road will always be composed of tarmac

Methods:

- run_quarter_mile(self,car)
    - This method takes in a car instance (of either ElectricCar or CombustionEngineCar) and outputs the amount of time (seconds) it takes to cross a quarter mile. The value it returns is a float data type and is rounded to two decimal points. Quarter mile time is determined by the car's wheel horsepower value.
    -
- run_0_60(self,car)
    - This method takes in a car instance (of either ElectricCar or CombustionEngineCar) and outputs the amount of time (seconds) it takes to reach 60 miles per hour. The value it returns is a float data type and rounded to two decimal points. 0 to 60 mph time is determined by the car's wheel horsepower value.

- race_cars(self,car_1,_car_2):
    - This method takes in two car instances (of either ElectricCar or CombustionEngineCar) and runs a "race" to see which car has a faster quarter mile time. It outputs the winner and loser of the race and provides their 0-60 mph and quarter mile times respectively